Разработване на тестове 2

In SystemVerilog there are two kinds of assertions: **immediate** (assert) and **concurrent** (assert property).

**Immediate Assertions**

Immediate assertions are procedural statements and are mainly used in simulation. An assertion is basically a statement that something must be true, similar to the if statement.

**assert** (A == B) $display ("OK. A equals B");

else $error("It's gone wrong");

**assert** (A == B) else $error("It's gone wrong");

**Concurrent Assertions**

The behaviour of a design may be specified using statements similar to these:

"The Read and Write signals should never be **asserted** together."

"A Request should be followed by an Acknowledge occurring no more than two clocks after the Request is asserted."

Concurrent assertions are used to check behaviour such as this. These are statements that assert that specified properties must be true. For example,

assert property (!(Read && Write));

asserts that the expression Read && Write is never true at any point during simulation.

NB! Not implemented in Vivado 2020

Properties are built using sequences. For example,

assert property (@(posedge Clock) Req |-> ##[1:2] Ack);

where

**Req** is a simple sequence (it’s just a boolean expression) and

**##[1:2] Ack** is a more complex sequence expression, meaning that Ack is true on the next clock, or on the one following (or both).

**|->** is the implication operator,

so this assertion checks that whenever Req is asserted, Ack must be asserted on the next clock, or the following clock.

Concurrent assertions like these are checked throughout simulation. They usually appear outside any initial or always blocks in modules, interfaces and programs.

The first assertion example above does not contain a clock. Therefore it is checked at every point in the simulation. // Not implemented in Vivado 2020

The second assertion is only checked when a rising clock edge has occurred; the values of Req and Ack are sampled on the rising edge of Clock.

**Пример**

<https://github.com/vtchoumatchenko/PCIS/tree/master/examples/timer-switch>

module timer\_switch\_top\_test;

localparam T = 10;

logic clock\_1Hz = 0;

logic reset, btn\_ext, light;

timer\_switch\_top uut(.\*);

always #(T/2) clock\_1Hz = ~clock\_1Hz;

initial begin

btn\_ext = 0;

reset = 0;

#T reset = 1;

#T reset = 0;

#T btn\_ext = 1;

#(4\*T) btn\_ext = 0;

#(30\*T) btn\_ext = 1;

#(5\*T) btn\_ext = 0;

#(10\*T) btn\_ext = 1;

#(3\*T) btn\_ext = 0;

#(30\*T) $finish;

end

// Check the after reset the ligh is off

assert property(@(posedge clock\_1Hz) reset |-> ~light);

// Check that:

// \* pressing the button turns on the light

// \* the light stays on for at least 20 seconds

assert property(@(posedge clock\_1Hz) ~btn\_ext ##1 btn\_ext |=> light [\*20]);

// equivalent to the previous property

assert property(@(posedge clock\_1Hz) **$rose**(btn\_ext) |=> light [\*20]);

endmodule

module timer\_switch\_test;

localparam T = 10;

logic clock = 0;

logic reset, btn, light;

timer\_switch uut(.clock\_1Hz(clock), .\*);

always #(T/2) clock = ~clock;

// Test with concurrent assertions

initial begin

btn = 0;

reset = 1;

#T reset = 0;

#T btn = 1;

#T btn = 0;

#(25\*T) btn = 1;

#T btn = 0;

#(10\*T) btn = 1;

#(T) btn = 0;

#(30\*T) $finish();

end

// Check state after reset

assert property(@(posedge clock) reset |-> uut.state == uut.OFF and ~light);

// Pressing the button turns on the lights

assert property(@(posedge clock) $rose(btn) |=> light);

// Check ON duration

property light\_is\_on\_for\_at\_least\_20cycles;

@(posedge clock) $rose(btn) |=> light [\*20];

endproperty;

assert property (light\_is\_on\_for\_at\_least\_20cycles);

// Check ON duration - alternative

property light\_remains\_on\_exactly\_20cycles\_after\_the\_last\_btn\_rise;

@(posedge clock) disable iff (reset) $fell(light) |-> $past(btn, 21);

endproperty;

assert property (light\_remains\_on\_exactly\_20cycles\_after\_the\_last\_btn\_rise);

endmodule

**Пример**

<https://github.com/vtchoumatchenko/PCIS/tree/master/examples/traffic-light/src>

import tlight\_package::\*;

module tlight\_test;

localparam T = 10;

logic clock = 0;

logic reset;

tlight\_control\_t ns, we;

tlight\_v2 uut(.\*);

always #(T/2) clock = ~clock;

initial begin

reset = 1;

#T reset = 0;

#(100\*T) $finish;

end

// Check the initial state

property after\_reset\_state;

@(posedge clock) $fell(reset) |-> uut.state == uut.RESET;

endproperty;

ap\_after\_reset\_state: assert property(after\_reset\_state);

//

// Check state duration and next state transition

//

// state: WE\_READY\_TO\_GO duration: 3s

// transition: WE\_READY\_TO\_GO -> WE\_GO

property check\_WE\_READY\_TO\_GO\_duration;

@(posedge clock) uut.state == uut.RESET ##1 uut.state == uut.WE\_READY\_TO\_GO |-> uut.state==uut.WE\_READY\_TO\_GO [\*3] ##1 uut.state==uut.WE\_GO;

endproperty;

ap\_check\_WE\_READY\_TO\_GO\_duration: assert property(check\_WE\_READY\_TO\_GO\_duration);

// state: WE\_READY\_TO\_GO duration: 3s - alternative

// The system function $past returns the value of an expression in a previous clock cycle.

property check\_WE\_READY\_TO\_GO\_duration\_alt;

@(posedge clock) uut.state == uut.WE\_READY\_TO\_GO ##1 uut.state == uut.WE\_GO |-> $past(uut.state==uut.WE\_READY\_TO\_GO,2);

endproperty;

ap\_check\_WE\_READY\_TO\_GO\_duration\_alt: assert property(check\_WE\_READY\_TO\_GO\_duration\_alt);

// state: WE\_GO duration: 15s

// transition: WE\_GO -> WE\_PREPARE\_TO\_STOP

property check\_WE\_GO\_duration;

@(posedge clock) uut.state == uut.WE\_READY\_TO\_GO ##1 uut.state == uut.WE\_GO |-> uut.state==uut.WE\_GO [\*15] ##1 uut.state==uut.WE\_PREPARE\_TO\_STOP;

endproperty;

ap\_check\_WE\_GO\_duration: assert property(check\_WE\_GO\_duration);

// state: WE\_PREPARE\_TO\_STOP duration: 1s

// transition: WE\_PREPARE\_TO\_STOP -> NS\_READY\_TO\_GO

property check\_WE\_PREPARE\_TO\_STOP\_duration;

@(posedge clock) uut.state == uut.WE\_GO ##1 uut.state == uut.WE\_PREPARE\_TO\_STOP |-> uut.state==uut.WE\_PREPARE\_TO\_STOP ##1 uut.state==uut.NS\_READY\_TO\_GO;

endproperty;

ap\_check\_WE\_PREPARE\_TO\_STOP\_duration: assert property(check\_WE\_PREPARE\_TO\_STOP\_duration);

// state: NS\_READY\_TO\_GO duration: 3s

// transition NS\_READY\_TO\_GO -> NS\_GO

property check\_NS\_READY\_TO\_GO\_duration;

@(posedge clock) uut.state == uut.WE\_PREPARE\_TO\_STOP ##1 uut.state == uut.NS\_READY\_TO\_GO |-> uut.state==uut.NS\_READY\_TO\_GO [\*3] ##1 uut.state==uut.NS\_GO;

endproperty;

ap\_check\_NS\_READY\_TO\_GO\_duration: assert property(check\_NS\_READY\_TO\_GO\_duration);

// state: NS\_GO duration: 15s

// transition NS\_GO -> NS\_PREPARE\_TO\_STOP

property check\_NS\_GO\_duration;

@(posedge clock) uut.state == uut.NS\_READY\_TO\_GO ##1 uut.state == uut.NS\_GO |-> uut.state==uut.NS\_GO [\*15] ##1 uut.state==uut.NS\_PREPARE\_TO\_STOP;

endproperty;

ap\_check\_NS\_GO\_duration: assert property(check\_NS\_GO\_duration);

// Verify state -> lights relations

ap\_check\_lights\_RESET: assert property

( @(posedge clock) uut.state==uut.RESET |-> ns==RED and we==RED);

ap\_check\_lights\_WE\_READY\_TO\_GO: assert property

( @(posedge clock) uut.state==uut.WE\_READY\_TO\_GO |-> ns==RED and we==YELLOW);

ap\_check\_lights\_WE\_GO: assert property

( @(posedge clock) uut.state==uut.WE\_GO |-> ns==RED and we==GREEN);

ap\_check\_lights\_WE\_PREPARE\_TO\_STOP: assert property

( @(posedge clock) uut.state==uut.WE\_PREPARE\_TO\_STOP |-> ns==RED and we==YELLOW);

ap\_check\_lights\_NS\_READY\_TO\_GO: assert property

( @(posedge clock) uut.state==uut.NS\_READY\_TO\_GO |-> ns==YELLOW and we==RED);

ap\_check\_lights\_NS\_GO: assert property

( @(posedge clock) uut.state==uut.NS\_GO |-> ns==GREEN and we==RED);

initial

begin

$display("%20s%6s%7s%7s","STATE","TIMER","WE","NS");

$monitor("%20s%6d%7s%7s", uut.state.name, uut.timer, ns.name, we.name);

end

endmodule